
Мониторинг черных
ящиков и котов
в мешке через eBPF
Пётр Бобров
HL
HighLoad++
2022
QIWI



Пётр Бобров

он же Дядя Петя

Специализация

- ВМК МГУ ' 1999
- 20 лет в ДБА
- 8.5 лет в ORACLE
- В QIWI — менеджер по отказоустойчивости
- Управляю бэклогом команды Site Reliabilty Engineering





ПО других поставщиков



Автоматизированная банковская система



Процессинг банковских карт

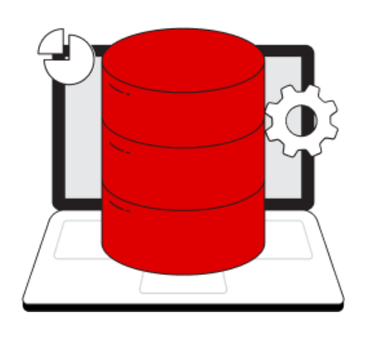


Системы дистанционного обслуживания





Системное ПО других поставщиков



Системы Управления Базами Данных



Операционные Системы





Проблемы мониторинга стороннего ПО

Отсутствие API и интерфейсов для метрик

Беспорядочное логирование

Лицензионные ограничения





История про карточный процессинг

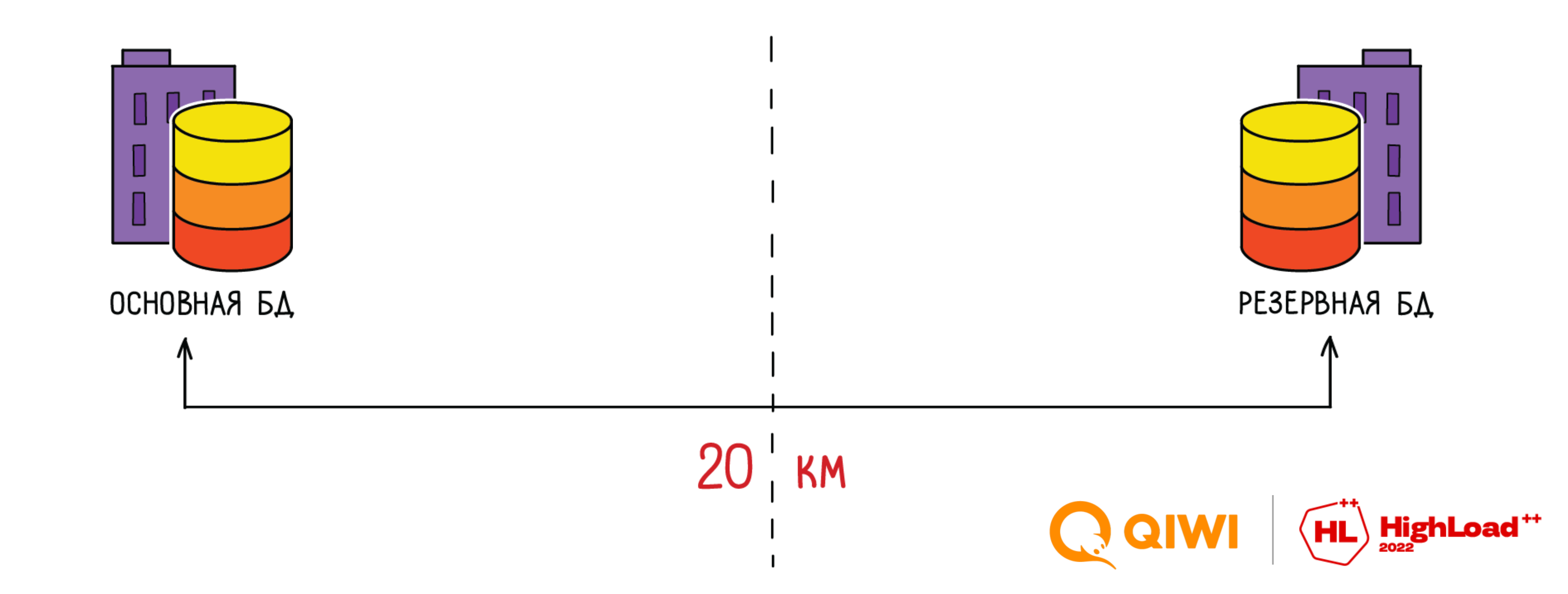
СУБД работает
в режиме синхронного коммита
для защиты от потери данных
при различных сбоях

За синхронный коммит
расплачиваемся задержками
обработки

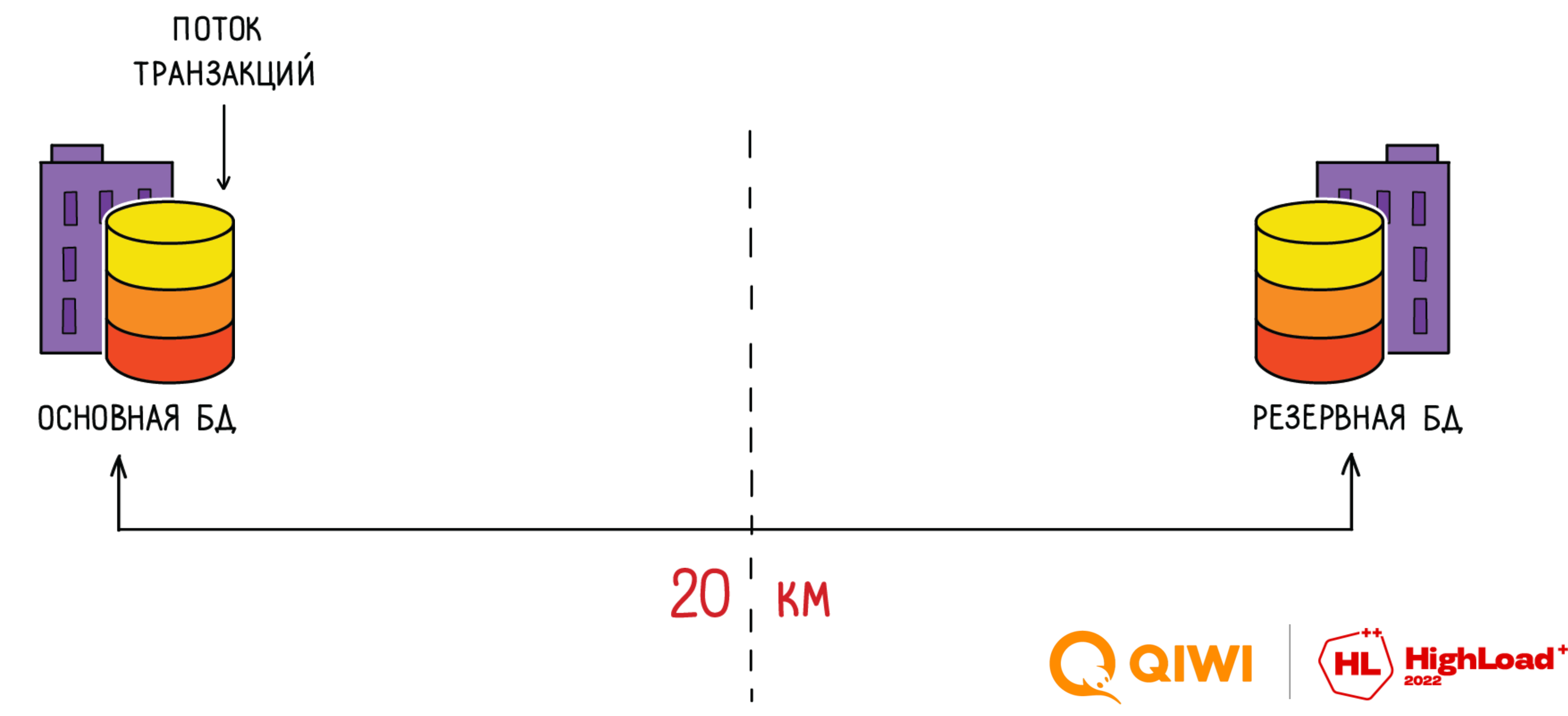




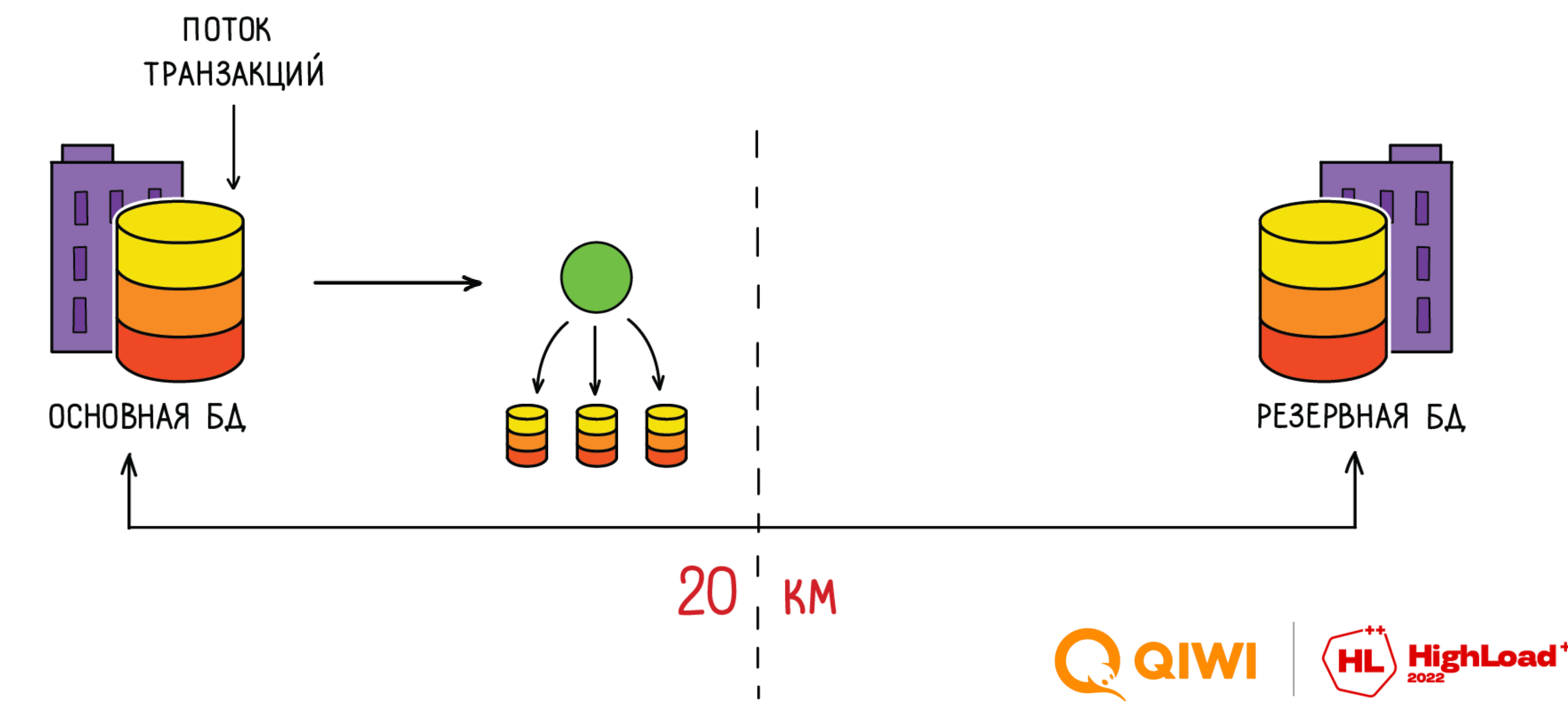
Архитектура синхронного коммита Oracle в карточном процессинге



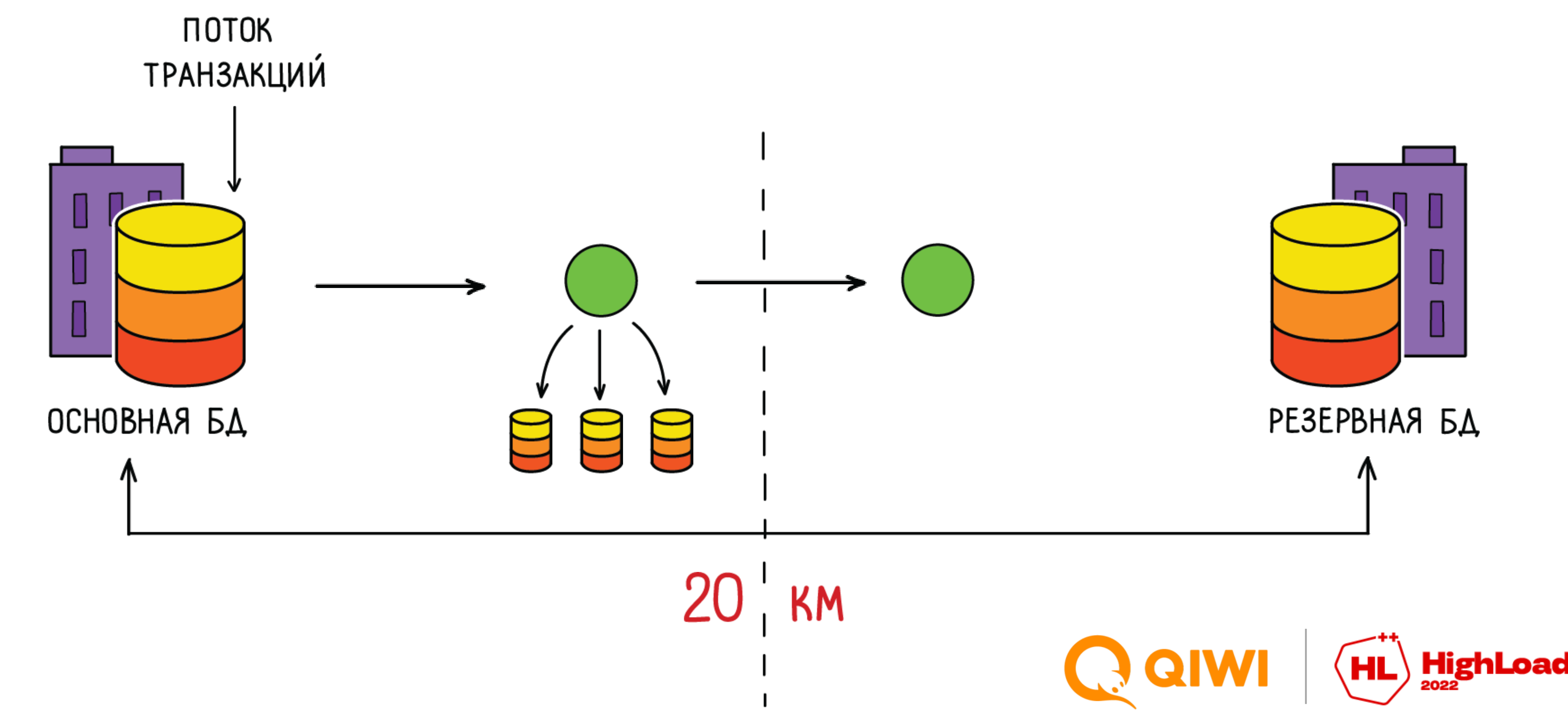
Архитектура синхронного коммита Oracle в карточном процессинге



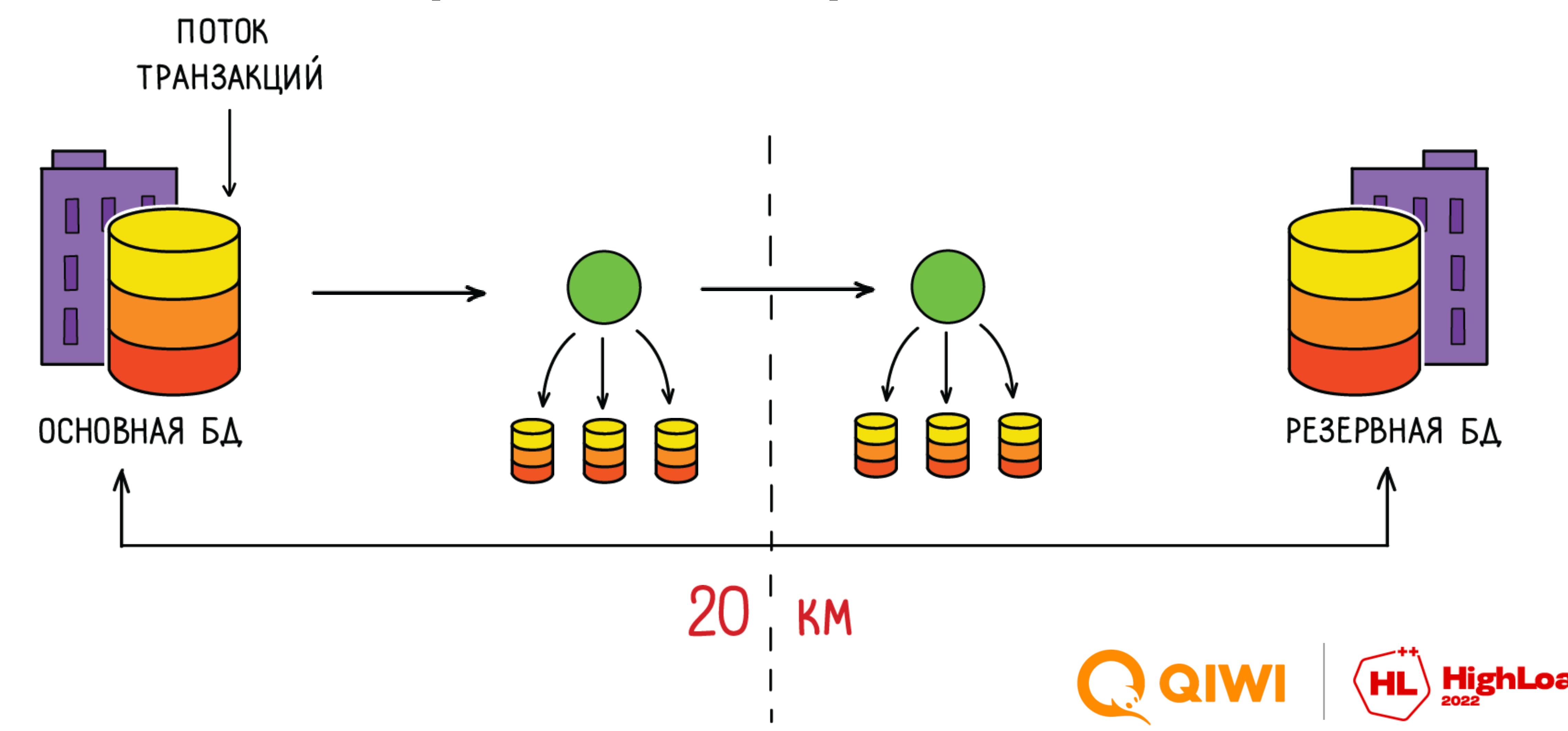
Архитектура синхронного коммита Oracle в карточном процессинге



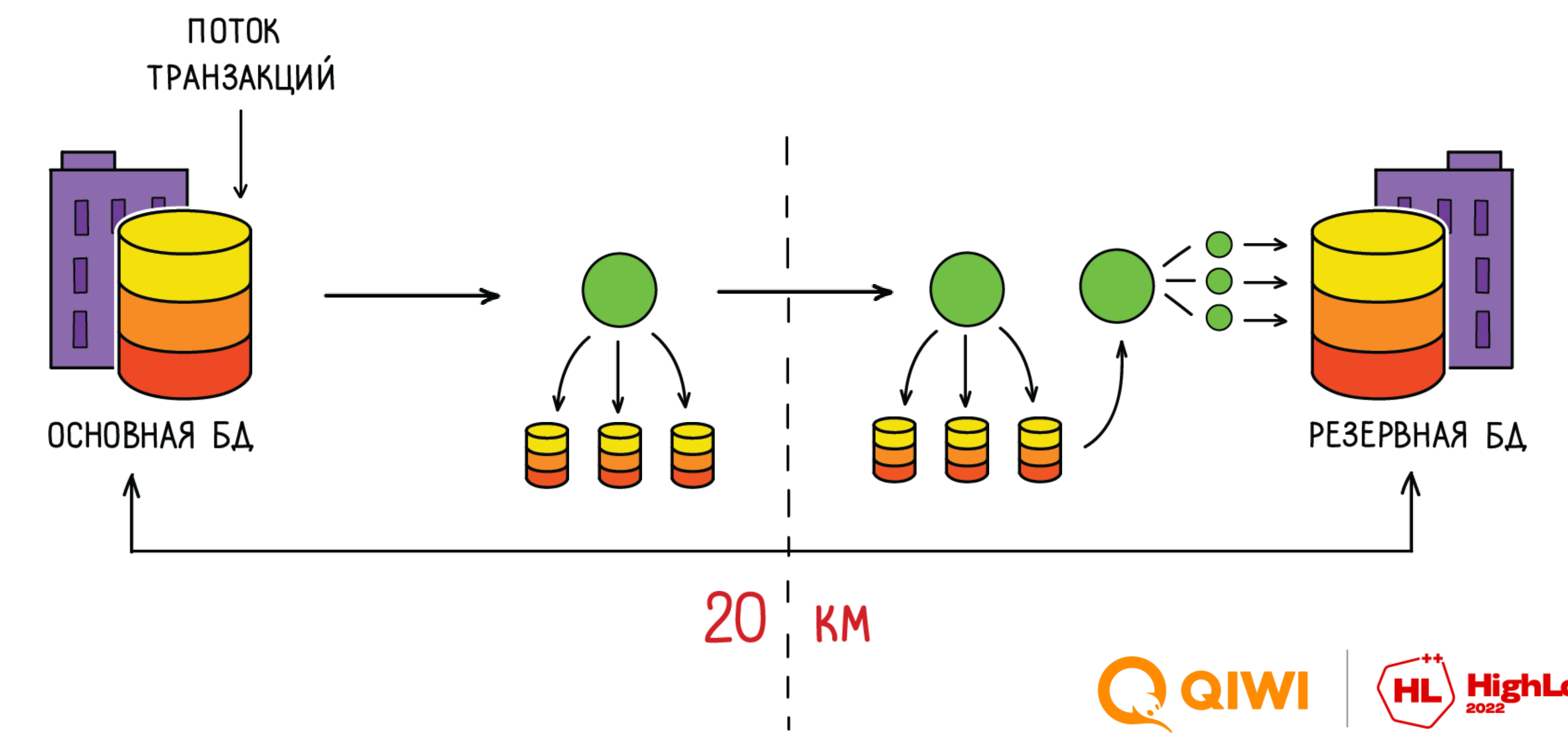
Архитектура синхронного коммита Oracle в карточном процессинге



Архитектура синхронного коммита Oracle в карточном процессинге



Архитектура синхронного коммита Oracle в карточном процессинге



Какие решения можно принять

SYNC

ASYNC

ОСНОВНАЯ БД

РЕЗЕРВНАЯ БД

РЕЗЕРВНАЯ БД





Какие решения можно принять

ASYNC

ASYNC

ОСНОВНАЯ БД

РЕЗЕРВНАЯ БД

РЕЗЕРВНАЯ БД





Какие решения можно принять

SYNC

РЕЗЕРВНАЯ БД

ОСНОВНАЯ БД

ASYNC

РЕЗЕРВНАЯ БД





Какие решения можно принять

ASYNC

РЕЗЕРВНАЯ БД

ОСНОВНАЯ БД

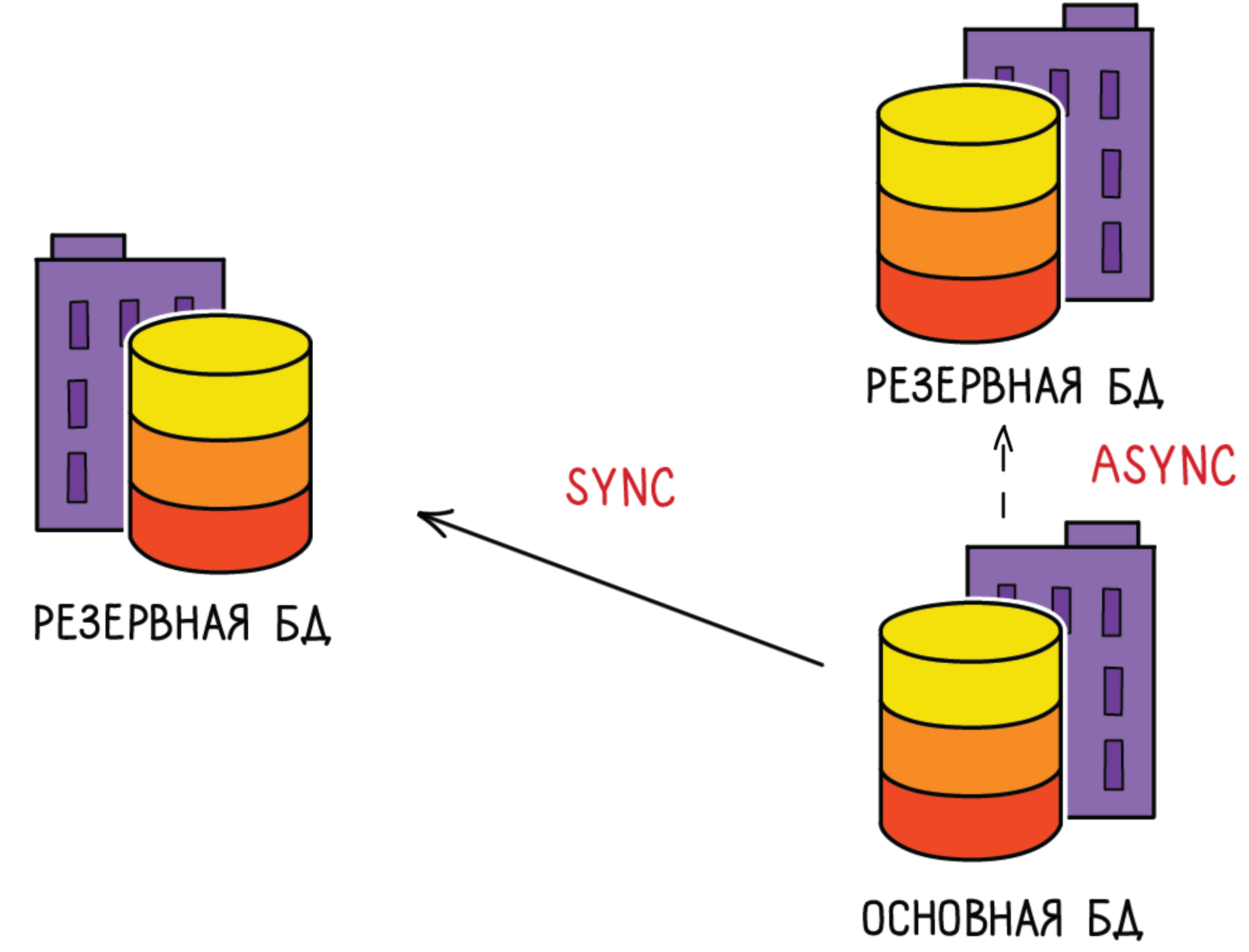
SYNC

РЕЗЕРВНАЯ БД





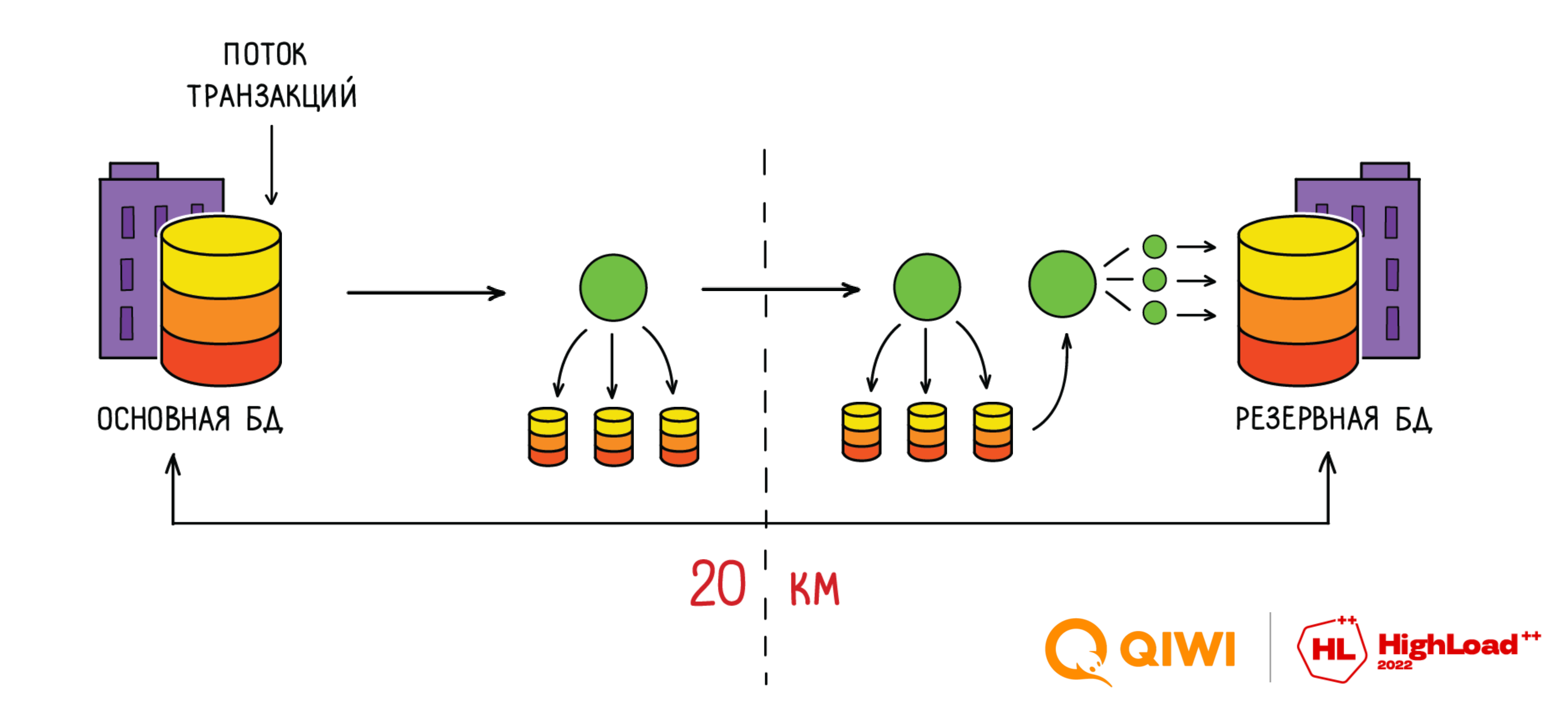
Какие решения можно принять



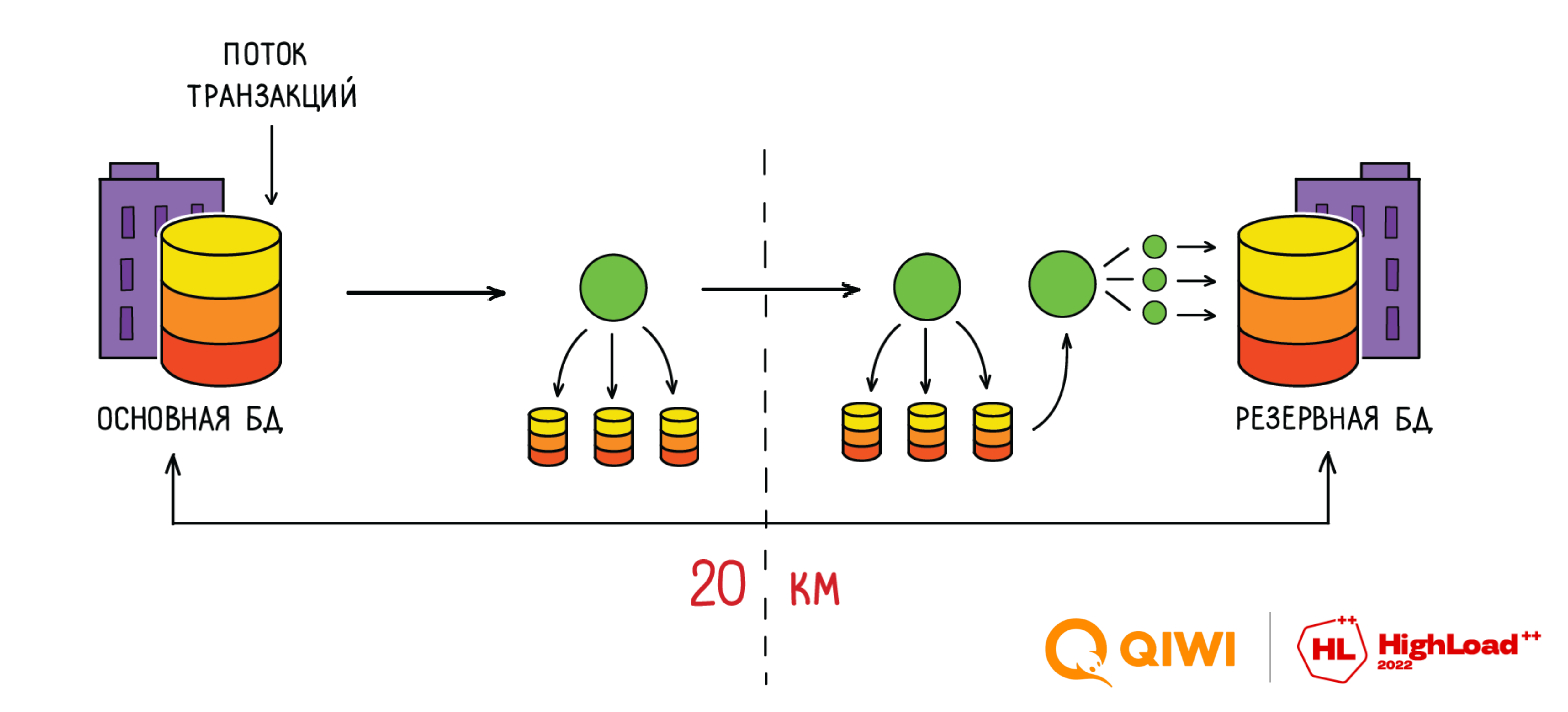




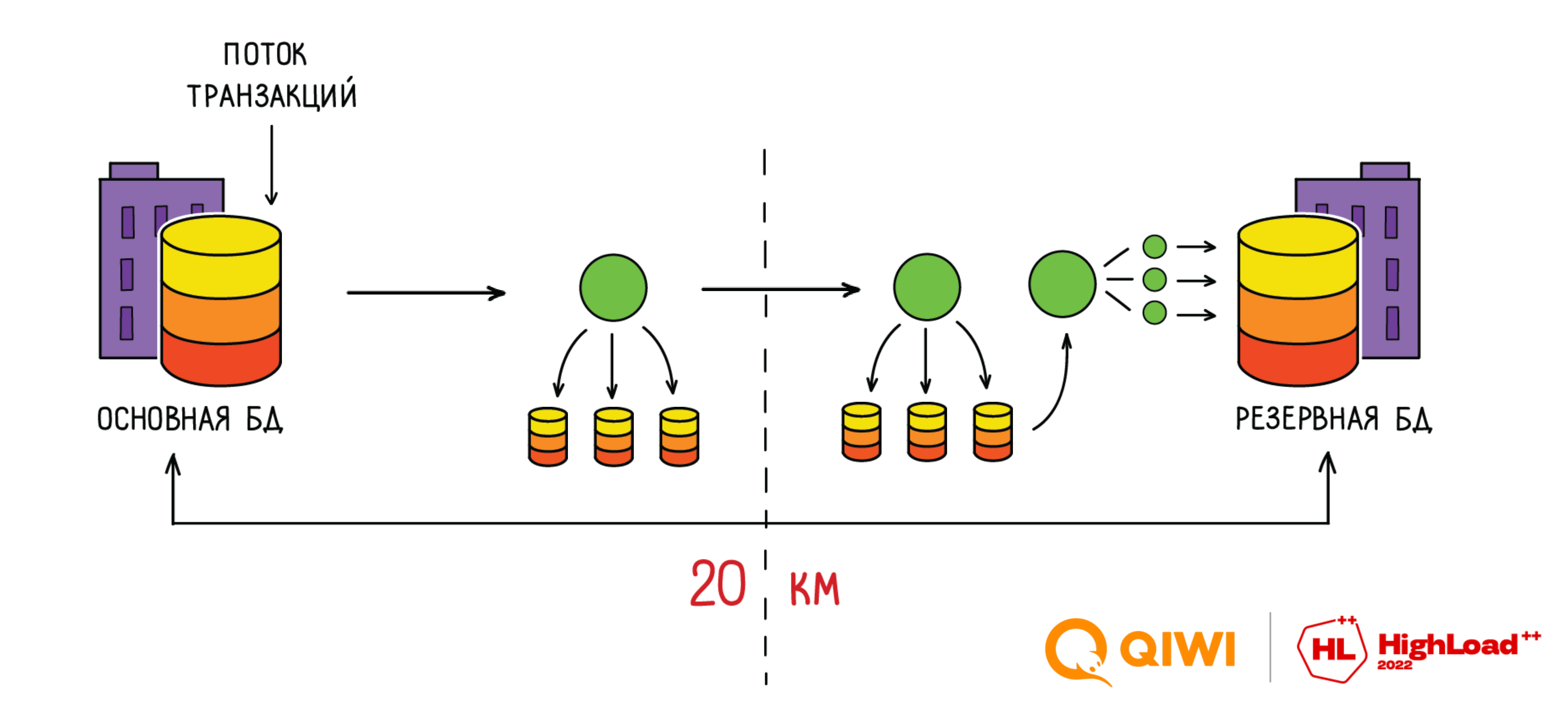
Что влияет на принятие решения



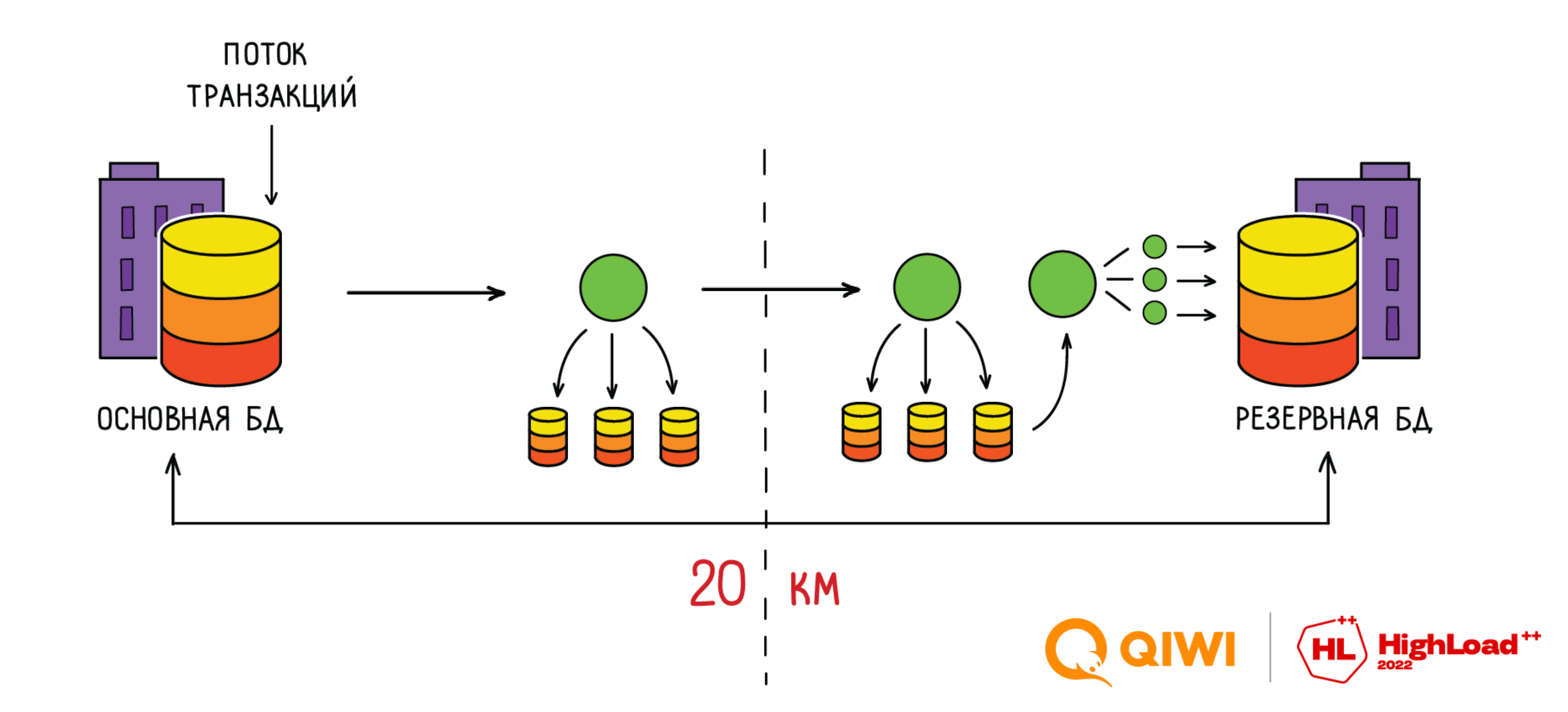
Что влияет на принятие решения



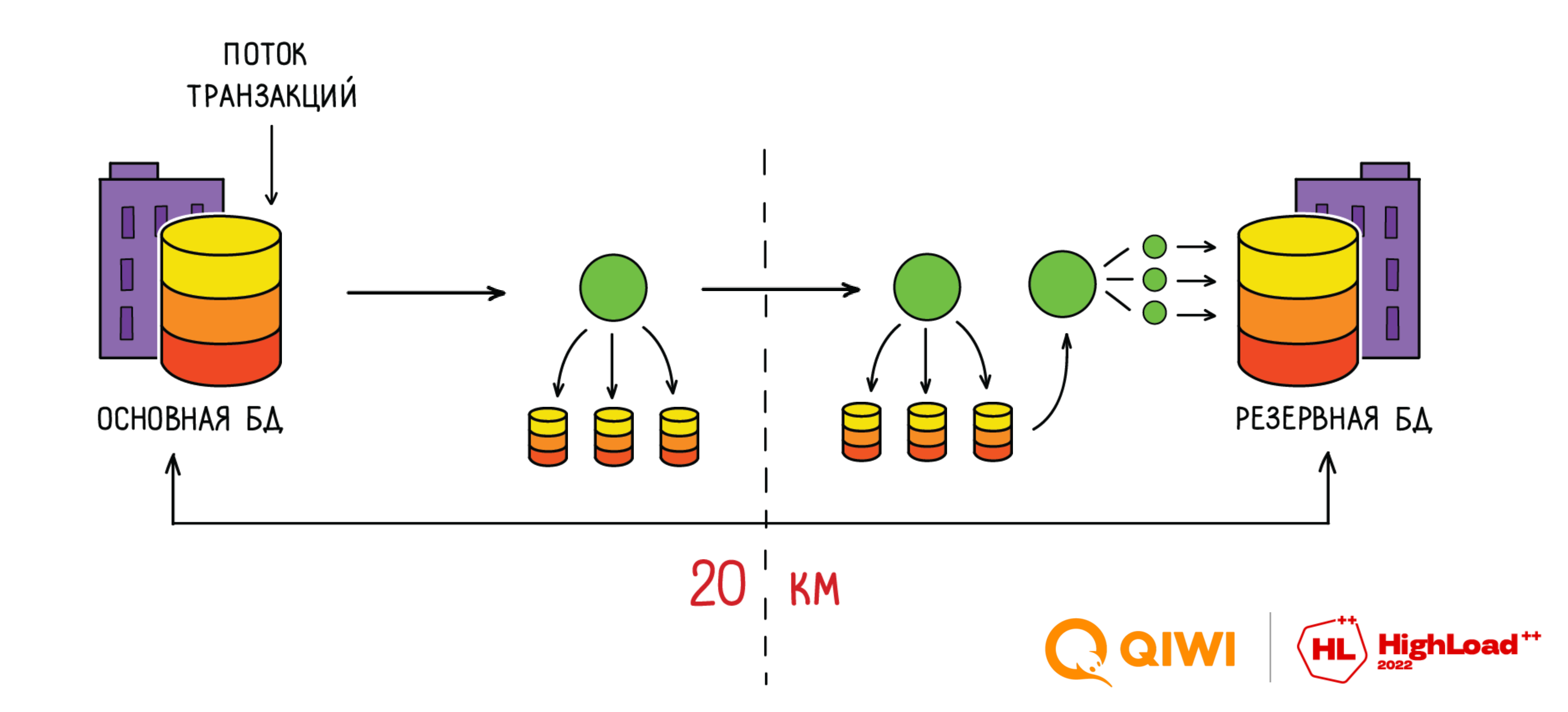
Что влияет на принятие решения



Что влияет на принятие решения



Что влияет на принятие решения



Средняя температура по больнице

Disk io_time

0.600 ms
0.500 ms
0.400 ms
0.300 ms
0.200 ms
0.100 ms
0 ms

	max	avg	current
dm-6	0.188 ms	0.104 ms	0.149 ms
dm-144	0.309 ms	0.123 ms	0.149 ms
dm-145	0.363 ms	0.138 ms	0.133 ms
dm-66	0.222 ms	0.110 ms	0.124 ms
dm-109	0.213 ms	0.107 ms	0.107 ms
dm-110	0.170 ms	0.101 ms	0.107 ms
dm-67	0.342 ms	0.134 ms	0.100 ms
dm-68	0.218 ms	0.111 ms	0.0987 ms
dm-146	0.202 ms	0.101 ms	0.0945 ms
dm-111	0.240 ms	0.106 ms	0.0898 ms
dm-112	0.237 ms	0.100 ms	0.0881 ms
dm-115	0.296 ms	0.0935 ms	0.0807 ms
dm-118	0.279 ms	0.0789 ms	0.0723 ms
dm-34	0.279 ms	0.0803 ms	0.0714 ms
dm-82	0.286 ms	0.0762 ms	0.0711 ms
dm-28	0.291 ms	0.0799 ms	0.0698 ms





Средняя температура по больнице

$$\frac{1\ \text{мс} * 10000 + 1\ \text{с} * 10}{10000 + 10} = \sim 2\ \text{мс}$$





История про карточный процессинг

Хочу знать длительность каждой дисковой операции и каждой сетевой операции





Что такое eBPF

Подсистема eBPF выполняет в ядре Linux пользовательский код защищенным способом





Было





Стало





Начало eBPF



index : kernel/git/torvalds/linux.git

Linux kernel source tree

about summary refs log tree **commit** diff stats

author	Alexei Starovoitov <ast@plumgrid.com>	2014-09-26 00:16:57 -0700
committer	David S. Miller <davem@davemloft.net>	2014-09-26 15:05:14 -0400
commit	99c55f7d47c0dc6fc64729f37bf435abf43f4c60 (patch)	
tree	12f09f26bee9813ae33cfc195582c41e94b2e4e9	
parent	4a8e320c929991c9480a7b936512c57ea02d87b2 (diff)	
download	linux-99c55f7d47c0dc6fc64729f37bf435abf43f4c60.tar.gz	

bpf: introduce BPF syscall and maps

```
BPF syscall is a multiplexor for a range of different operations on eBPF.
This patch introduces syscall with single command to create a map.
Next patch adds commands to access maps.

'maps' is a generic storage of different types for sharing data between kernel
and userspace.

Userspace example:
/* this syscall wrapper creates a map with given type and attributes
 * and returns map_fd on success.
 * use close(map_fd) to delete the map
 */
int bpf_create_map(enum bpf_map_type map_type, int key_size,
                   int value_size, int max_entries)
{
```





Виртуальная машина eBPF

01

Виртуальный RISC-процессор

Свой собственный набор команд, включая загрузку/выгрузку памяти, функции, сравнения, условные переходы и исключая циклы

Описан в исходниках ядра в bpf.h

02

Имеет стек и 11 64х-битных регистров

512 байт стека.

Часть регистров передаются между программами как параметры функций, часть — только для чтения

03

В ней запрещена прямая адресация памяти

Регистры могут указывать на стек, но не на прямые адреса памяти.

Работа с фиксированными структурами памяти — контекст, maps и т д. — входят в набор команд машины.





eBPF-программы — похожи на триггеры в БД, но вызываются на какое-то событие в Linux

01	**Это, вообще, безопасно?**	eBPF-код проверяется специальным верифаером
02	**Будет ли оверхед?**	Ограничения на программу гарантируют приемлемый оверхед.
03	**Как обращаться к результатам?**	Структуры памяти жестко заданы заранее, eBPF-программа их заполняет, пользовательский процесс их читает





Основные точки вызова eBPF-программ

01

Kernel probes
Вызов любой разрешенной функции в ядре, получение ее параметров и результата

02

User probes
Вызов любой функции в пользовательской программе, получение ее параметров и результата

03

Tracepoints
Вызов функции, специально помеченной разработчиком как трассировочный вызов

04

Perf events
Доступ к счетчикам CPU и ядра Linux





Как это работает: в указанной точке появляется инструкция int 3

Обычный код

```
Dump of assembler code for function:
0x000000000048aea0 <+0> :        push  %r12
0x000000000048aea2 <+2> :        push  %rbp
0x000000000048aea3 <+3> :        push  %rbx
0x000000000048aea4 <+4> :        add   $0xffffffffffffff80 , %rsp
0x000000000048aea8 <+8> :        mov   %fs:0x28, %rax
0x000000000048aeb1 <+17> :   mov   %rax, 0x78(%rsp)
0x000000000048aeb6 <+22> :   xor   %eax, %eax
```

Прицепим eBPF:

```
Dump of assembler code for function:
0x000000000048aea0 <+0> :        int 3
0x000000000048aea0 <+1> :        push  %rsp
0x000000000048aea2 <+2> :        push  %rbp
0x000000000048aea3 <+3> :        push  %rbx
0x000000000048aea4 <+4> :        add   $0xffffffffffffff80 , %rsp
0x000000000048aea8 <+8> :        mov   %fs:0x28, %rax
0x000000000048aeb1 <+17> :   mov   %rax, 0x78(%rsp)
0x000000000048aeb6 <+22> :   xor   %eax, %eax
```





Главные операционные риски

01

Безопасность

Уязвимости в ядре компрометируют сразу всю систему – поэтому рекомендую разрешать eBPF исключительно для root

02

Совместимость

Новые версии Linux могут не работать со старыми версиями скриптов и наоборот.

03

Баги

Случалось, что аттач eBPF- программы к uprobe крэшил пользовательский процесс

04

Закладки

Скомпилировать из исходников подсистему eBPF достаточно сложно





Как сделать мониторинг, если я менеджер

HELP ME
STACK OVERFLOW
YOU'RE MY ONLY HOPE





План аналогичного проекта

01 Поиск подходящих точек для мониторинга

02 Поиск существующего eBPF-кода

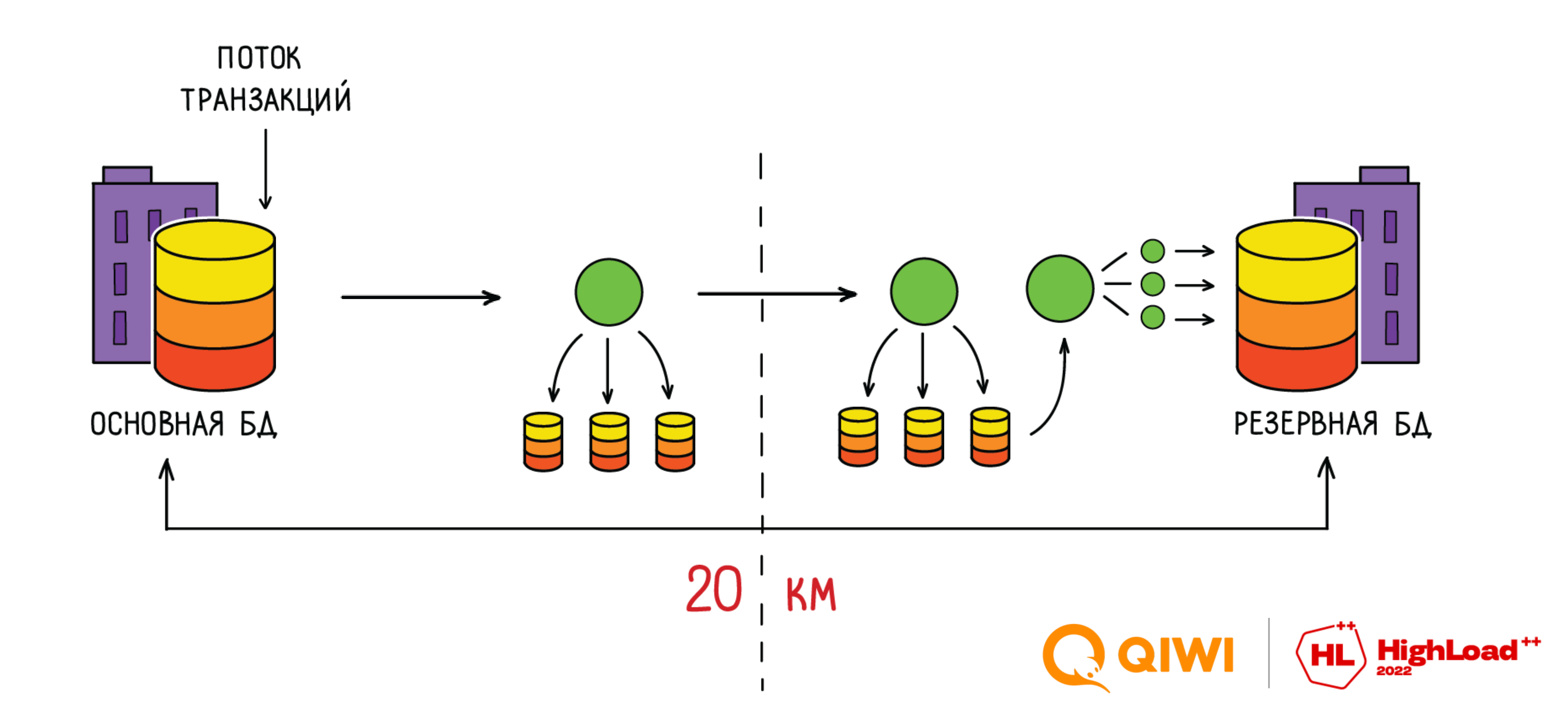
03 Proof of Concept

04 Интеграция с корпоративными системами

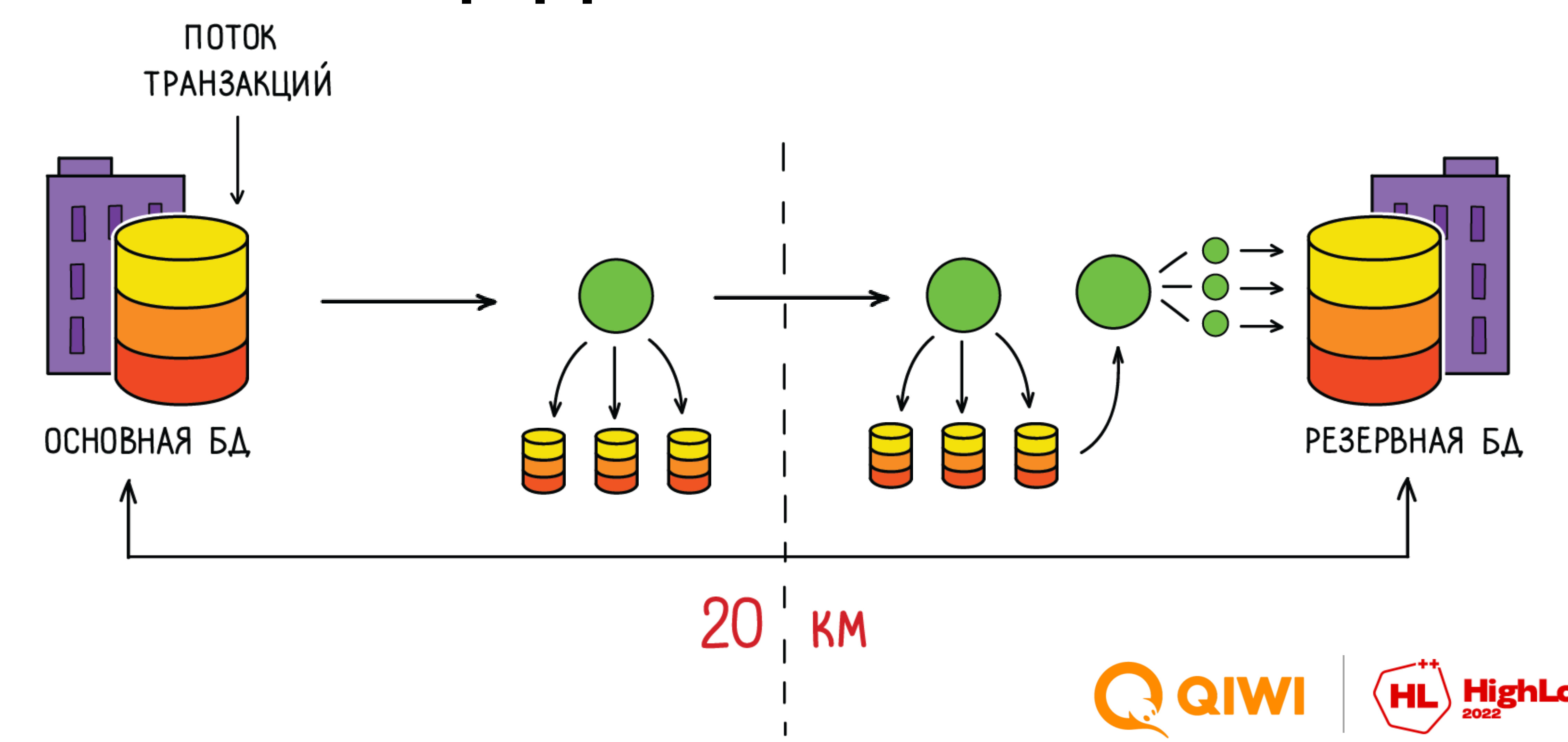




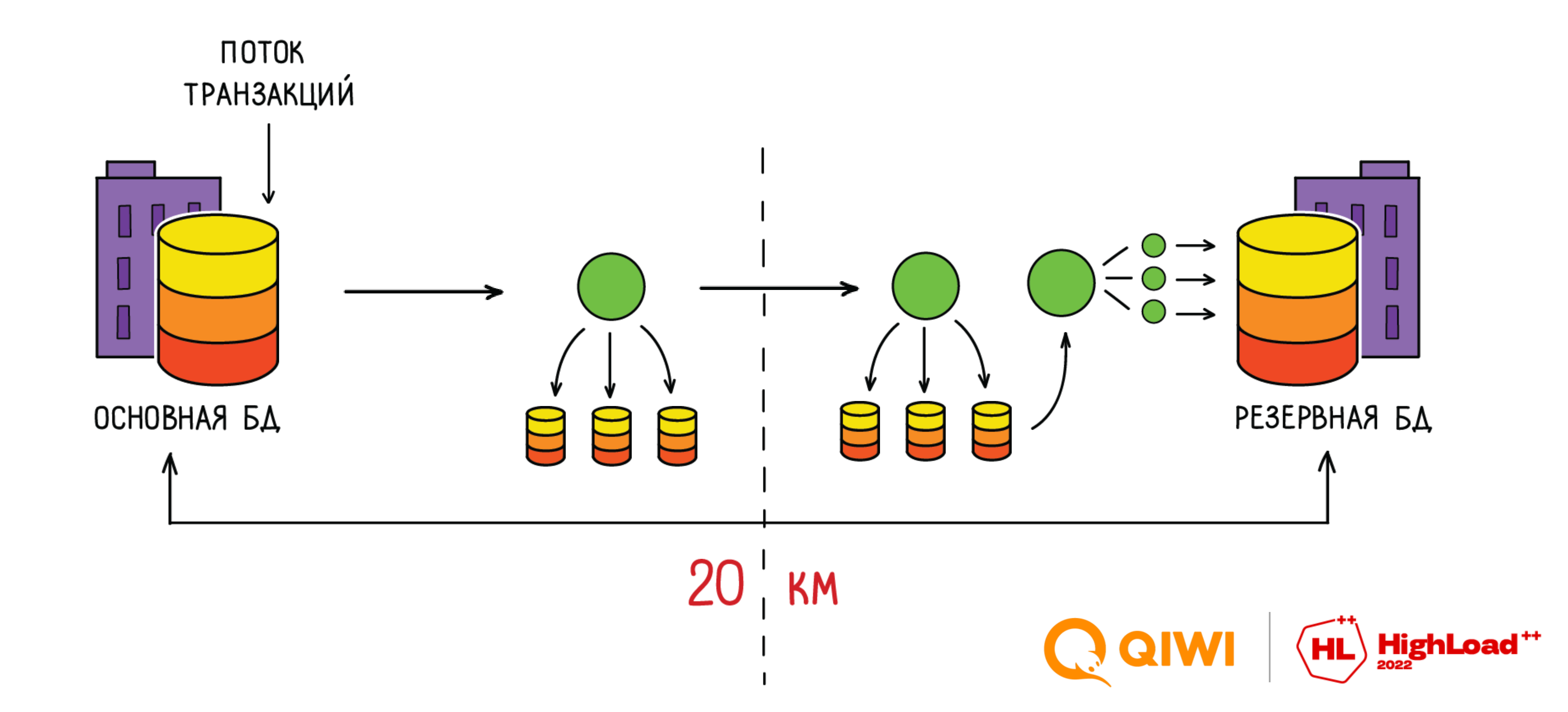
Поиск точек для мониторинга



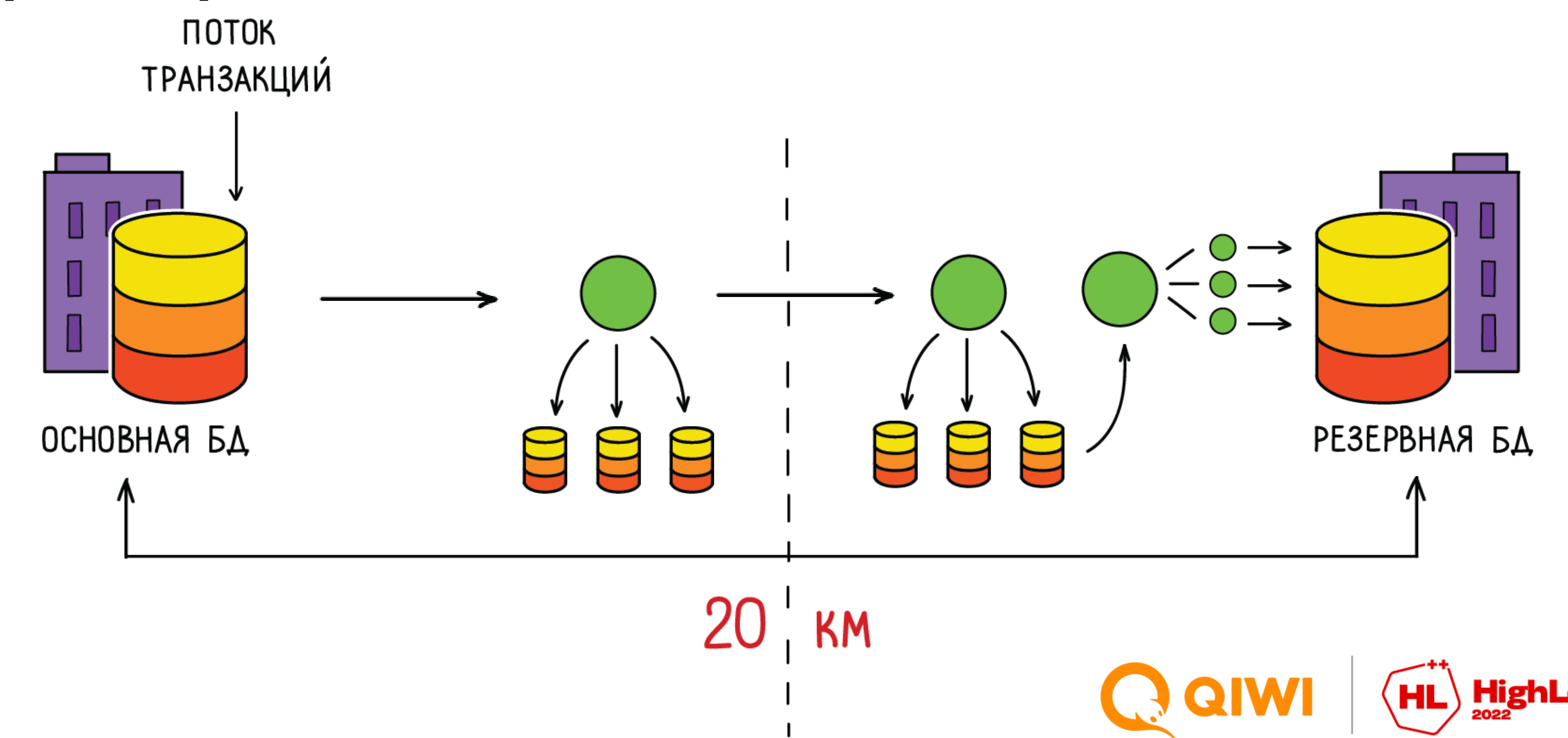
Событие записи блока на диск в основном ЦОД



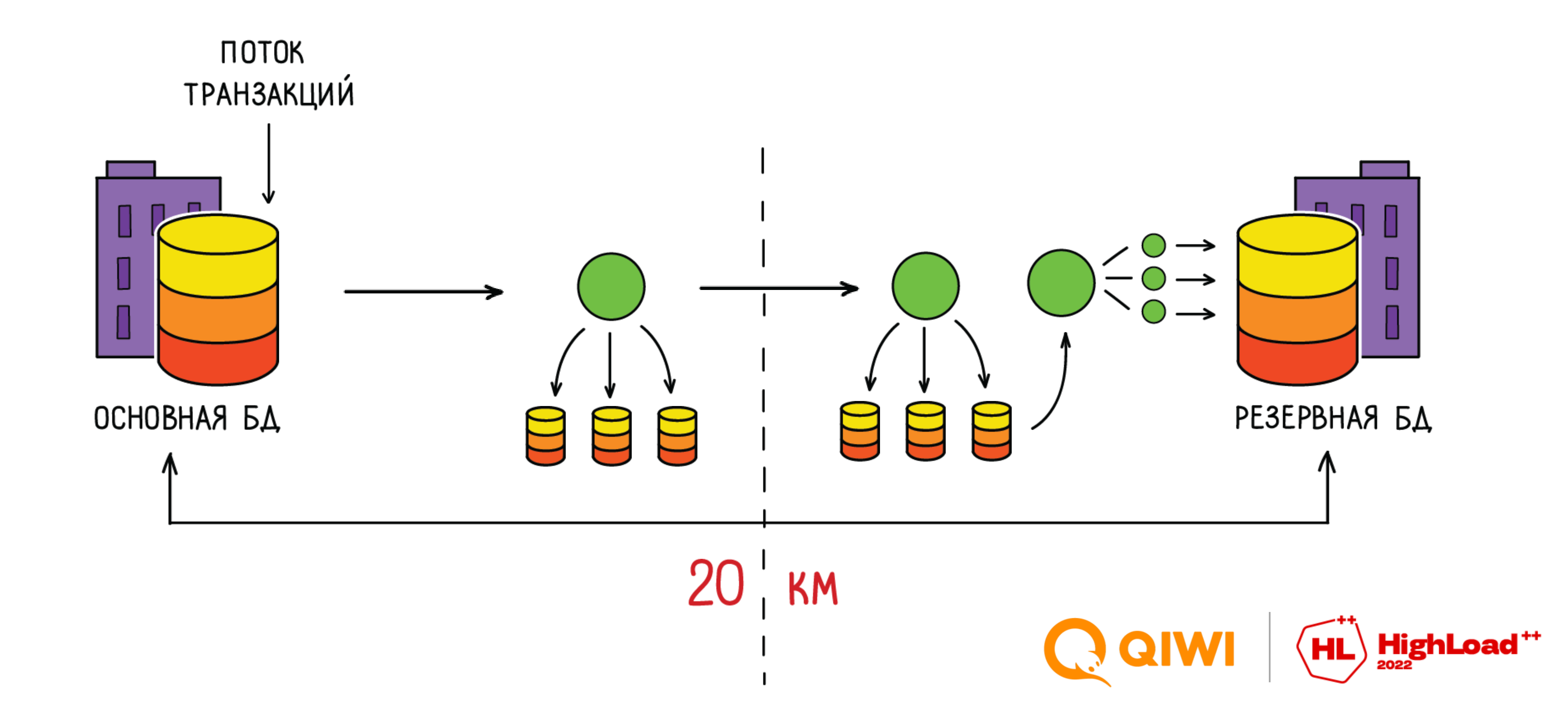
Событие передачи пакета по сети



Событие записи блока на диск в резервном ЦОД



Хочу измерять все эти события!



Коллекция готовых программ BCC

iovisor.github.io/bcc/



BCC

Dynamic Tracing Tools for Linux





Коллекция готовых программ BCC

Disk I/O Latency Histogram

```
# biolatency
Tracing block device I/O... Hit Ctrl-C to end.
^C
     usecs               : count     distribution
       0 -> 1          : 0        |                                       |
       2 -> 3          : 0        |                                       |
       4 -> 7          : 0        |                                       |
       8 -> 15         : 0        |                                       |
      16 -> 31         : 0        |                                       |
      32 -> 63         : 0        |                                       |
      64 -> 127        : 1        |                                       |
     128 -> 255        : 12       |********                               |
     256 -> 511        : 15       |**********                             |
     512 -> 1023       : 43       |*******************************        |
    1024 -> 2047       : 52       |**************************************|
    2048 -> 4095       : 47       |**********************************     |
    4096 -> 8191       : 52       |**************************************|
    8192 -> 16383      : 36       |**************************             |
   16384 -> 32767      : 15       |**********                             |
   32768 -> 65535      : 2        |*                                      |
   65536 -> 131071     : 2        |*                                      |
```





eBPF-программа: области памяти

github.com/iovisor/bcc/blob/master/tools/biolatency.py

```
# define BPF program                # ← eBPF-программа на С пишется в текстовую переменную Python
bpf_text = """

#include <uapi/linux/ptrace.h>
#include <linux/blk-mq.h>

typedef struct disk_key {
    char disk[DISK_NAME_LEN];
    u64 slot;
} disk_key_t;
…
BPF_HASH(start, struct request *);  # ← определяется структура памяти для внутреннего использования
…
```





eBPF-программа: собственно, код

github.com/iovisor/bcc/blob/master/tools/biolatency.py

```
...
STORAGE
int trace_req_start(struct pt_regs *ctx, struct request *req)          # ← тут первая функция eBPF-кода
{
u64 ts = bpf_ktime_get_ns();
    start.update(&req, &ts);
    return 0;
}
int trace_req_done(struct pt_regs *ctx, struct request *req)           # ← тут вторая функция eBPF-кода
{
    u64 *tsp, delta;
    // fetch timestamp and calculate delta
    tsp = start.lookup(&req);
    if (tsp == 0) {
        return 0;   // missed issue
    }
...
```





eBPF-программа: собственно, код 2

github.com/iovisor/bcc/blob/master/tools/biolatency.py

```
…
delta = bpf_ktime_get_ns() - *tsp;
    …
    // store as histogram
    STORE
    start.delete(&req);
    return 0;
}
"""

…
storage_str += "BPF_HISTOGRAM(dist, disk_key_t);"          # ← определение "внешней" структуры памяти
store_str += "dist.atomic_increment(bpf_log2l(delta));“    # ← и ее заполнение результатом работы

bpf_text = bpf_text.replace("STORAGE", storage_str)
bpf_text = bpf_text.replace("STORE", store_str)
…                                                          # ← тут сама eBPF-программа на С заканчивается
```





eBPF-программа: аттач к событию

github.com/iovisor/bcc/blob/master/tools/biolatency.py

```
...
# load BPF program
b = BPF(text=bpf_text)                                          # ← тут eBPF-программа компилируется
if args.queued:
    if BPF.get_kprobe_functions(b'__blk_account_io_start'):
        b.attach_kprobe(event="__blk_account_io_start", fn_name="trace_req_start")      # ← аттач к функции
    else:
        b.attach_kprobe(event="blk_account_io_start", fn_name="trace_req_start")
else:
    if BPF.get_kprobe_functions(b'blk_start_request'):
        b.attach_kprobe(event="blk_start_request", fn_name="trace_req_start")
    b.attach_kprobe(event="blk_mq_start_request", fn_name="trace_req_start")
if BPF.get_kprobe_functions(b'__blk_account_io_done’):
    b.attach_kprobe(event="__blk_account_io_done", fn_name="trace_req_done")
else:
    b.attach_kprobe(event="blk_account_io_done", fn_name="trace_req_done")
...
```





eBPF-программа: вывод результата

github.com/iovisor/bcc/blob/master/tools/biolatency.py

```
dist = b.get_table("dist")            # ← тут читаем "внешнюю", заполненную программой, область памяти
...
while (1):
    try:
        sleep(int(args.interval))
    except KeyboardInterrupt:
        exiting = 1
...
    dist.print_log2_hist(label, "disk", disk_print)          # ← тут выводим отформатированный результат
...
    dist.clear()
...
countdown -= 1
    if exiting or countdown == 0:
        exit()
```





Итак, я нашёл то, что мне нужно

github.com/iovisor/bcc/blob/master/tools/biolatency.py

```
# biolatency
Tracing block device I/O... Hit Ctrl-C to end.
^C
     usecs               : count     distribution
         0 -> 1          : 0        |                                      |
         2 -> 3          : 0        |                                      |
         4 -> 7          : 0        |                                      |
         8 -> 15         : 0        |                                      |
        16 -> 31         : 0        |                                      |
        32 -> 63         : 0        |                                      |
        64 -> 127        : 1        |                                      |
       128 -> 255        : 12       |********                              |
       256 -> 511        : 15       |**********                            |
       512 -> 1023       : 43       |*******************************       |
      1024 -> 2047       : 52       |**************************************|
      2048 -> 4095       : 47       |**********************************    |
      4096 -> 8191       : 52       |**************************************|
      8192 -> 16383      : 36       |**************************            |
     16384 -> 32767      : 15       |**********                            |
     32768 -> 65535      : 2        |*                                     |
     65536 -> 131071     : 2        |*                                     |
```





Что мне надо изменить в этом коде?

Мне надо поменять функцию print так, чтобы вместо вывода в консоль сформировать строку по правилам сервиса telegraf и отправлять эту строку в сервис telegraf по протоколу http





Вопросы по Python, решенные поиском

- Как вывести в буфер вместо stdout
- Как добавить метод к классу в другом файле – «monkey patching»
- Как написать функцию
- Как записать буфер по http
- Как разобрать IP-адрес





Мой первый open-source:

github.com/unPeter/bcc2telegraf







Как мне запускать программу автоматически на всех серверах?

Мне надо создать Linux-сервис, управляемый systemctl, написать для него модуль в puppet, замерджить в репозиторий и массово распространить.





Вопросы по shell-скриптам, puppet и systemctl, решенные поиском

- Как написать в Linux systemd-managed-сервис
- Как правильно написать модуль в puppet
- Как правильно написать в yaml многострочные параметры для hiera
- Как выводить результаты в лог-файл, не блокируя доступ к нему
- Как правильно пользоваться git в рабочем процессе, в основном git fetch -- all && git reset --hard





Как мне получить красивые графики, по которым можно принимать решения?

Мне надо визуализировать сетевые и дисковые задержки с помощью HeatMap и показать такие HeatMap для всех ЦОД на одной странице





Как устроен HeatMap

2 s
1 s
511 ms
255 ms
127 ms
63 ms
31 ms
15 ms
7 ms
3 ms
1 ms

17:25 17:30 17:35 17:40 17:45 17:50 17:55 18:00 18:05 18:10 18:15 18:20

1 20,000 44,049





Вопросы по grafana, решенные поиском

- Как делать дропдаун-параметры в дашборде
- Как правильно сделать HeatMap
- Как правильно агрегировать измерения / метки на PromQL
- Какой еще информации не хватает на дашике
- Как правильно описать методику использования дашиков для принятия решений





Что получилось в конце концов



Обратная связь и комментарии по докладу по ссылке



